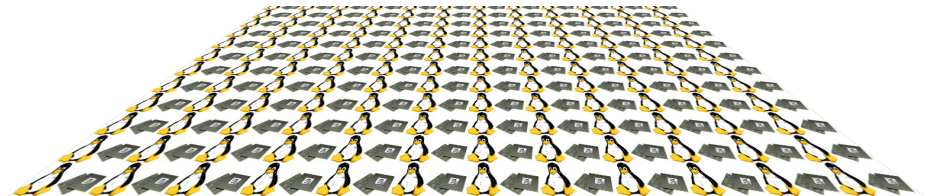
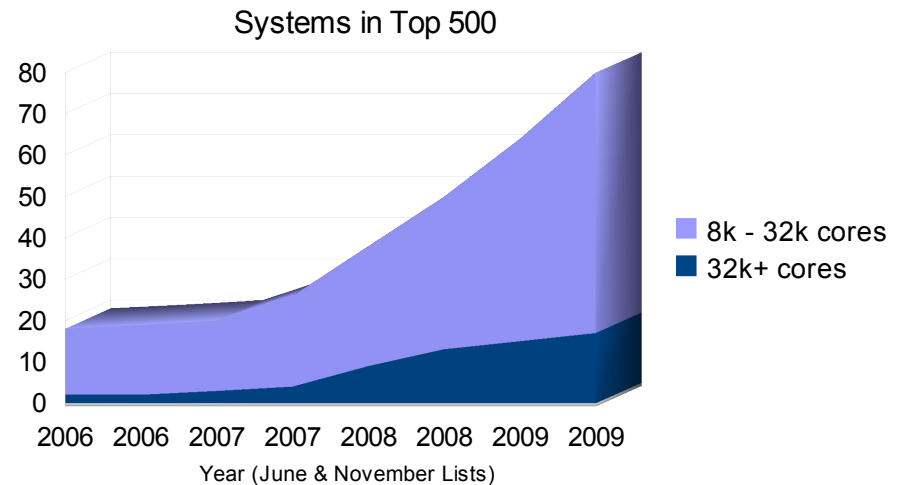




Development Tools for Multicore Systems

David Lecomber
david@allinea.com
CTO

- Processor counts growing rapidly
- GPUs entering HPC
- Large hybrid systems imminent
- But what happens when software doesn't work?



- HPC tools company since 2001
 - DDT - Debugger for MPI, threaded/OpenMP and scalar
 - OPT - Optimizing and profiling tool for MPI and non-MPI
 - DDTLite - Parallel Debugging Plugin for Microsoft Visual Studio 2008 SP1 and above
- Large European and US customer base
 - Ease of use - means tools get used
 - Users debugging regularly at all scales
 - Scalable interface - easy to use at 1 or 100,000s of cores
- Looking to the future
 - In use at Petascale
 - GPU product in Beta



- Academic

- Over 200 universities



- Major research centres

- ANL, EPCC, IDRIS, Juelich, NERSC, ORNL,



- Aviation and Defense

- Airbus, AWE, Dassault, DLR, EADS, ...



- Energy

- CEA, CGG Veritas, IFP, Total, ..



- EDA

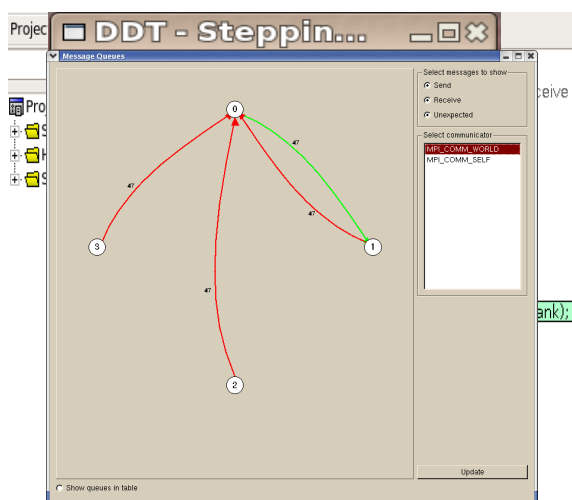
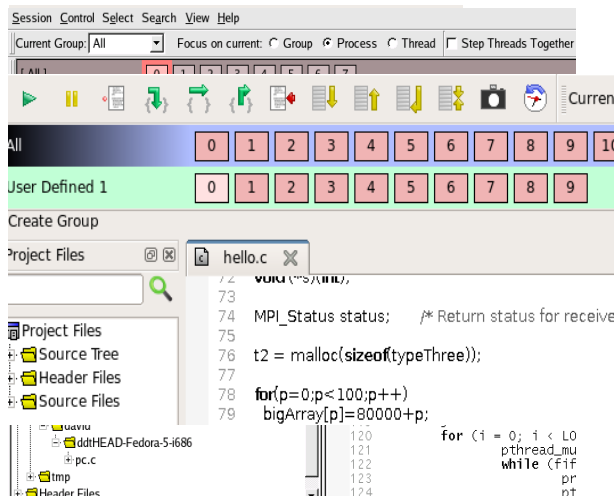
- Cadence, Intel, Synopsys, ...



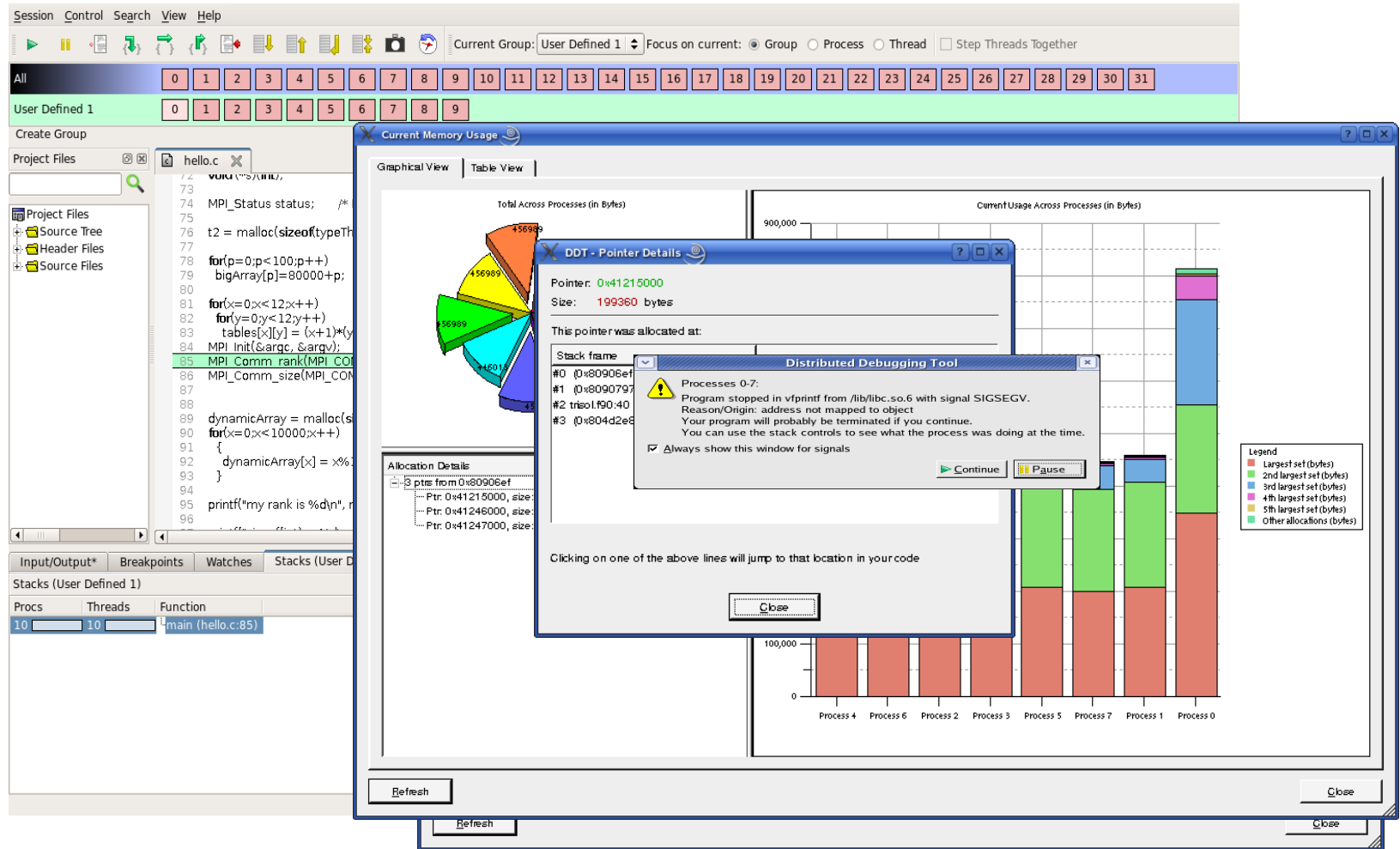
- Climate and Weather

- UK Met Office, Meteo France, ...

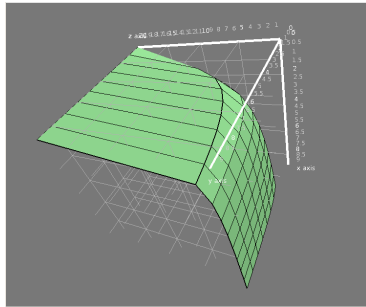
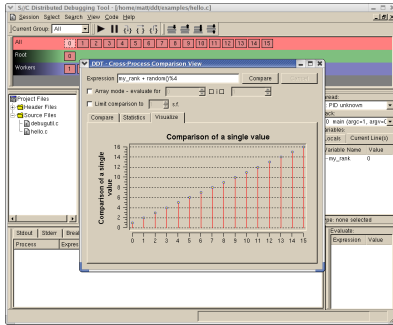
- A powerful and highly intuitive tool
 - Traditional focus has been HPC
- Cross-platform support
 - Linux, Solaris, AIX, Super UX, Blue Gene O/S
 - Blue Gene, Cell, x86-64, ia64, PowerPC, Sparc, NEC, NVIDIA
 - GNU, Absoft, IBM, Intel, PGI, Pathscale, Sun compilers
- Across all MPI and OpenMP implementations
 - From low end to high end
- Support for all scheduling systems
 - SGE, PBS, LSF, MOAB, ...
 - Flexible, powerful, easy to use queue submission



- Scalar features
 - Advanced C++ and STL
 - Fortran 90, 95 and 2003: modules, allocatable data, pointers, derived types
 - Memory debugging
- Multithreading & OpenMP features
 - Step, breakpoint etc. one or all threads
- MPI features
 - Easy to manage groups
 - Control processes by groups
 - Compare data
 - Visualize message queues



- Cross process/thread comparison
- Visualize multidimensional data



Group Expression: array[0][0]

Range of 0: 0 to 15, Range of 1: 0 to 15, Range of 2: 0 to 15, Range of 3: 0 to 15

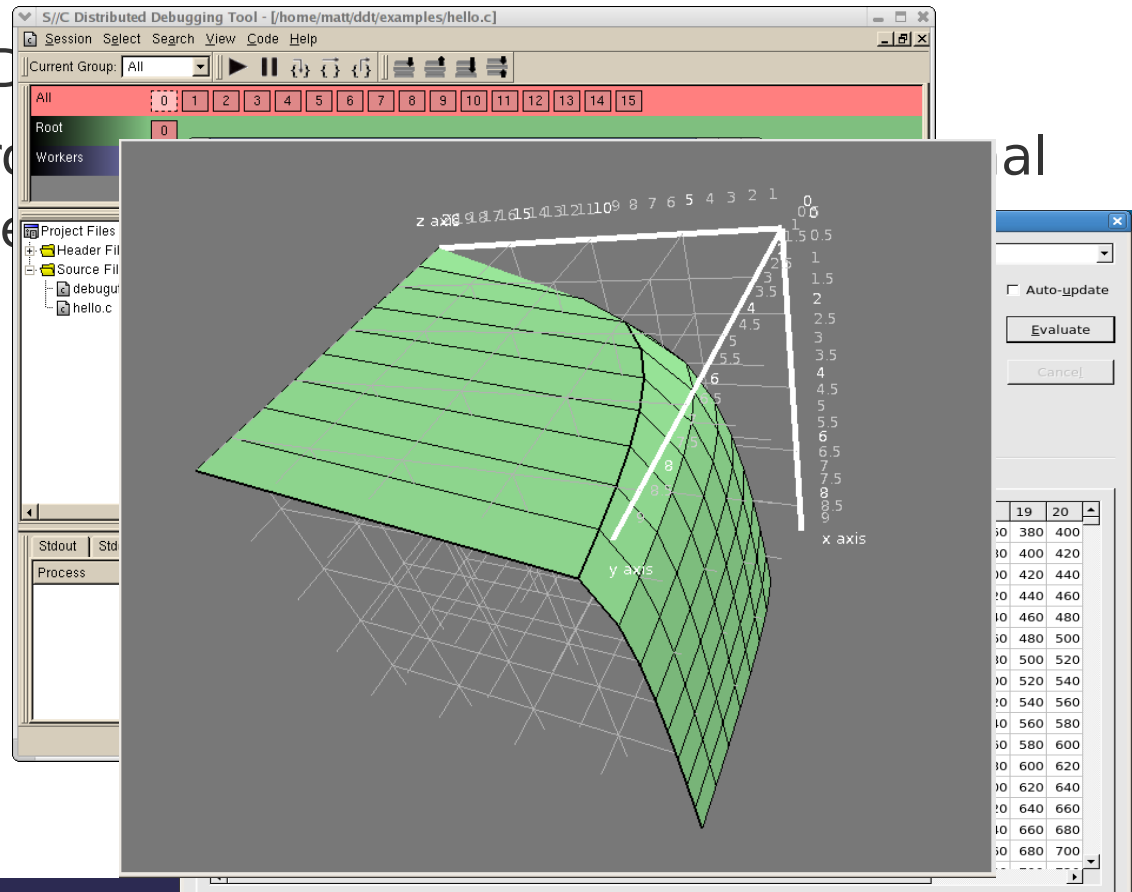
Display: From, Display: From, Display: From, Display: Columns

Approximate function: None

Expression: "array[0][0]" evaluated for process 0 at 11:49

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
10	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
11	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
12	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
13	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
14	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
15	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

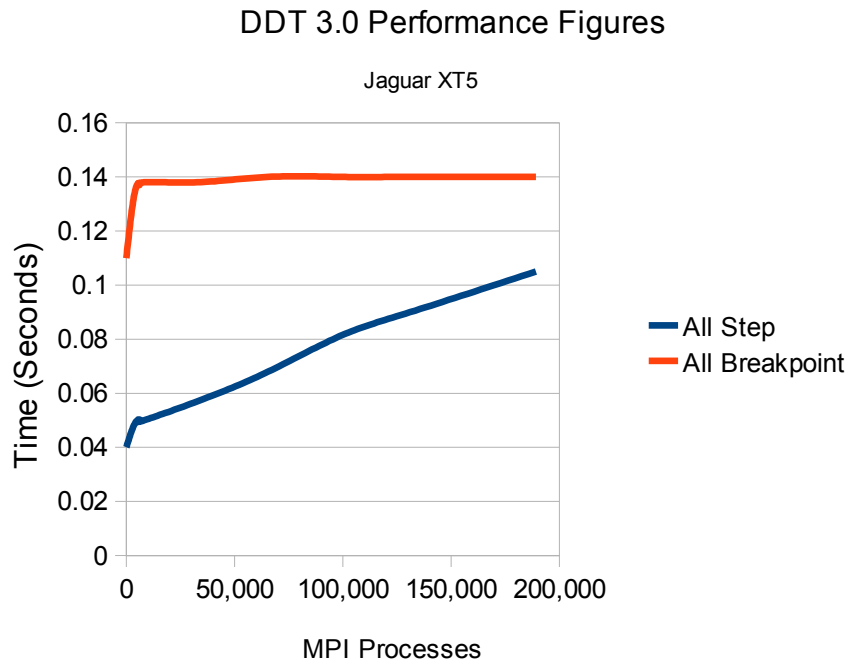
– 3D
– Fro
vie

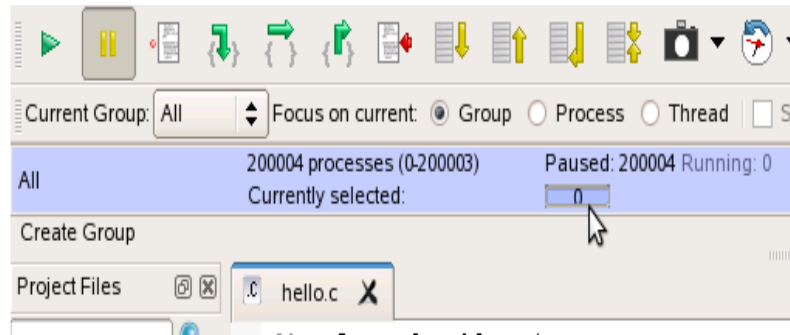


Expression "iarray[0][0]" evaluated for process 0 at 11:49.

Visualize in 3D Export to Spreadsheet... Close

- DDT is delivering petascale debugging **today**
 - Collaboration with ORNL on Jaguar Cray XT
 - Tree architecture – logarithmic performance
 - Many operations now faster at 220,000 than previously at 1,000 cores
 - **~1/10th of a second** to step and gather all stacks at 220,000 cores

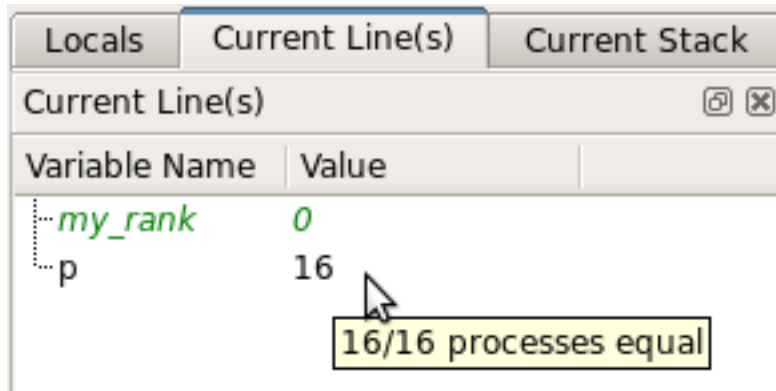




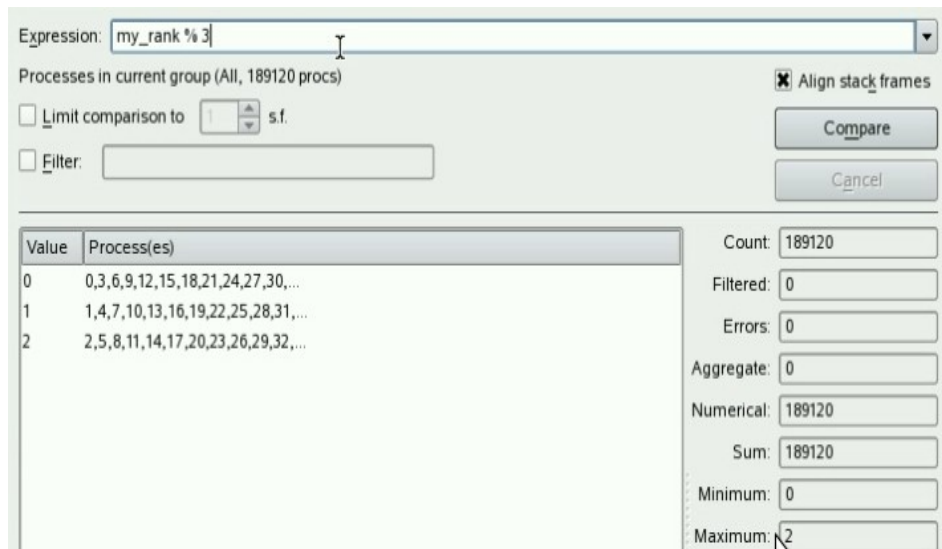
- Control Processes by Groups
 - Set breakpoints, step, play, stop etc. using user-defined groups
 - Scalable process groups view
 - Compact representation
- Parallel Stack View
 - Finds rogue processes faster
 - Identifies classes of process behaviour
 - Allows rapid grouping of processes

A screenshot of the 'Stacks (All)' view in the Allinea SPC interface. It displays a list of processes and their current function. The processes are grouped by address (150120, 150119, and 1). The functions shown include _start, __libc_start_main, main, pop (POP.f90:81), initialize_pop (initial.f90:119), init_communicate (communicate.f90:87), create_ocn_communicator (communicate.f90:300), and create_ocn_communicator (communicate.f90:303).

Processes	Function
150120	_start
150120	__libc_start_main
150120	main
150120	pop (POP.f90:81)
150120	initialize_pop (initial.f90:119)
150120	init_communicate (communicate.f90:87)
150119	create_ocn_communicator (communicate.f90:300)
1	create_ocn_communicator (communicate.f90:303)



- Gather from every node
 - Potentially costly – if all data different
 - Easy if data mostly same
 - New ideas
 - Aggregated statistics
 - Probabilistic algorithms optimize performance – even in pathological case
 - ~130ms for 130,000 cores
- Watch this space!
 - With a fast and scalable architecture, new things become possible



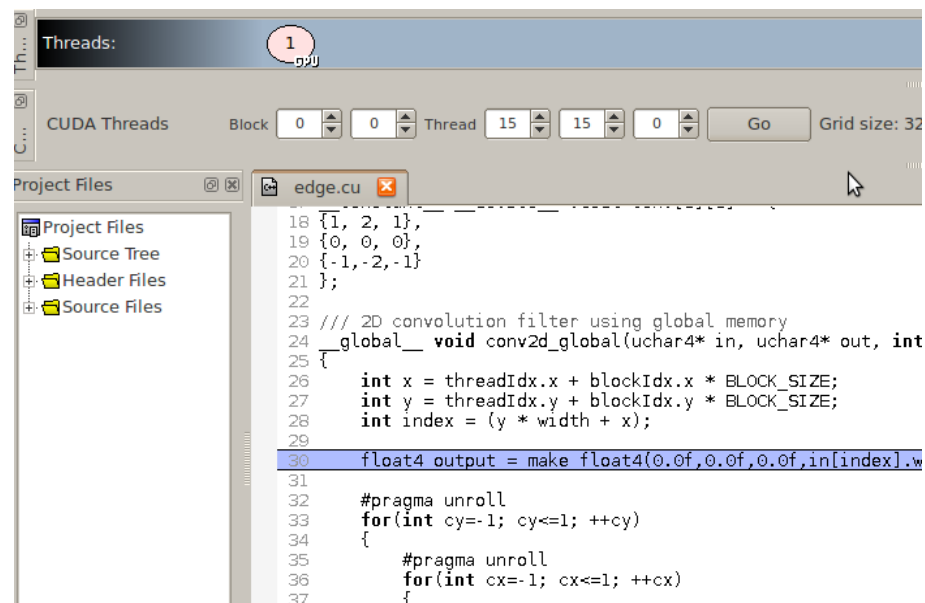
- DDT is the first Petascale debugger..
 - A debugging tool has finally caught up with the hardware!
 - Work is in progress to port every feature for scale
 - Memory debugging, data visualization,
 - How can the infrastructure be built upon?
 - Does DDT offer the right framework for collaboration?
 - Can we encourage a codebase of user-generated MPI tools/utilities?
- ... but large clusters are a fraction of HPC
 - Most parallel development starts smaller
 - Is now starting even smaller: GPUs

- Dominant technology is Linux clusters
 - Not fast enough? Add another rack.
 - Still not fast enough? Buy a better network.
 - *Still* not fast enough? Wait six months and buy another system.
 - ... and then the electric bill arrives
- *Easy* to use
 - Vast collection of existing codes: compile and go.
- Good ecosystem of development tools
 - Compiler support: codes port easily between systems
 - Debugging tools and optimization tools – eg. DDT and OPT
 - Easy to use and common interface across many system types

- Hybrids are a hot topic
 - Technology is moving quickly – compilers, SDKs, hardware
 - CUDA currently at the front in tool support
- Many lines of code need rewriting for GPUs
 - Memory hierarchy
 - Explicit data transfer between host and accelerator
 - Unusual execution model -
 - Kernels, thread blocks, warps, synchronization points
 - Do developers really know how their code is executed?
 - Massively parallel model
 - Single pass in a for-loop is the new granularity

- Old world “printf”
 - NVIDIA SDK 3.0 allows this (new) – but has limitations
- Fake it – run the kernel on the host x86_64 processor
 - Languages often support targeting host CPU instead of GPU
 - Different numeric precision – different answer?
 - Different scheduling – different answer?
 - A reasonable option for some bugs
- Run on the GPU with Allinea DDT
 - Very close collaboration with the NVIDIA debugger team
 - In use by early access customers – requires NVIDIA SDK 3.0
 - Release of public beta – awaiting imminent SDK 3.0 release

- Run the code
 - Browse source
 - Set breakpoints
 - Stop at a line of CUDA code
 - Stops once for each scheduled collection of blocks
- Select a CUDA thread
 - Examine variables and shared memory
 - Step a warp
 - View all extant threads in parallel tree view



```
18 {1, 2, 1},
19 {0, 0, 0},
20 {-1, -2, -1}
21 };
22
23 /// 2D convolution filter using global memory
24 _global_ void conv2d_global(uchar4* in, uchar4* out, int
25 {
26     int x = threadIdx.x + blockIdx.x * BLOCK_SIZE;
27     int y = threadIdx.y + blockIdx.y * BLOCK_SIZE;
28     int index = (y * width + x);
29
30     float4 output = make_float4(0.0f, 0.0f, 0.0f, in[index].w);
31
32     #pragma unroll
33     for(int cy=-1; cy<=1; ++cy)
34     {
35         #pragma unroll
36         for(int cx=-1; cx<=1; ++cx)
37         {
```

Threads	Function
1	main (edge.cu:75)
32	conv2d_global (edge.cu:35)
480	conv2d_global (edge.cu:39)



- Threads
 - Scheduled in batches, short lifetime
 - Identified by thread index and block index
 - Each part of a warp (32 threads in a warp)
 - Local state and shared data
- Loop iteration to thread analogy?
 - Don't want to watch detail of every thread
 - But do want to pick some to check the logic
 - eg. start, end, and interior points

- Compile your code for debugging:
 - Just add “-g” flag during compilation
- DDT is installed on Jaguar, Franklin and Hopper
 - module load ddt
 - ddt
- That's all - you're debugging!